Model Checking - Métodos Formais

MILTON PEDRO PAGLIUSI NETO, UDESC, Brazil
EDUARDO PANDINI, UDESC, Brazil

Neste artigo referente ao trabalho da matéria de Métodos Formais, apresentaremos o desenvolvimento, usabilidade e conceito de Model Checking, um método formal ágil, utilizado para validação de diferentes propriedades em um sistema de modelos (que utilizam máquinas de estados finitas).

CCS Concepts: • **Formal Methods -> Model Checking**; • **Computational Theory**;

Additional Key Words and Phrases: model checking, formal methods, metodos formais, autômatos finitos

ACM Reference Format:
Milton Pedro Pagliusi Neto and Eduardo Pandini. 2022. Model Checking - Métodos Formais. 1, 1 (July 2022), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Considerando o atual cenário na área de informação, a necessidade de uma implementação ágil, com uma base de recursos concretos e alta confiabilidade, como sistemas financeiros, torna-se essencial a utilização de diversas ferramentas no processo de testes e análise de diferentes aplicações, e uma destas ferramentas chama-se *"Model Checking"*.

Tem como base os conceitos e fundações da lógica proposicional, linguagens formais, estrutura de dados, teoria de grafos, assim como também teoria dos autômatos, fazendo parte de métodos formais, *"Model Checking"* trata-se de uma técnica formal e automatizada (aplicada para modelar e analisar sistemas de TI) para verificação de um programa, checando a existência de erros ou violações de suas propriedades.

Podemos elaborar como exemplos destas propriedades, como a garantia da não-existência de um deadlock durante a execução do programa, significando que não deve haver um ponto no processamento que deixe o programa sem um caminho para continuar. Estas propriedades são implementadas através de uma linguagem formal, a fim de evitar a introdução de problemas durante essa especificação, como ambiguidade.

Temos como vantagens da aplicação de *"Model Checking"* a velocidade da verificação, diagnóstico de contra-exemplos e a lógica proposicional expressando diferentes propriedades concorrentes com facilidade.

Como desvantagem do uso de *"model checking"* é a explosão de estados, na mesma proporção em que a complexidade de um modelo aumenta, e os estados da máquina de estados aumenta exponencialmente, junto com a verificação destes.

Authors' addresses: Milton Pedro Pagliusi Neto, UDESC, Joinville, Brazil, milton-pedro@hotmail.com; Eduardo Pandini, UDESC, Joinville, Brazil, pandiniedu@gmail.com.

XXXX-XXXX/2022/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 VERIFICAÇÃO

O processo de verificação segue as seguintes etapas:

2.1 Modelagem

Nessa primeira etapa, é necessário elaborar um modelo meticulosamente para retratar a aplicação com o conjunto de estados identificando comportamento, e as arestas identificando a transição de estados, as especificações são feitas conforme o comportamento esperado do programa. [1]

2.2 Execução

Nesta etapa, um model checker é chamado, e verifica o modelo e se as condições esperadas são atingidas.

2.3 Análise

Nesta etapa, se a condição for atingida, a próxima condição é colocada para a verificação, caso falhe, o modelo pode ter engatilhado uma explosão de estados(complexo demais), ou a condição está empregada incorretamente.

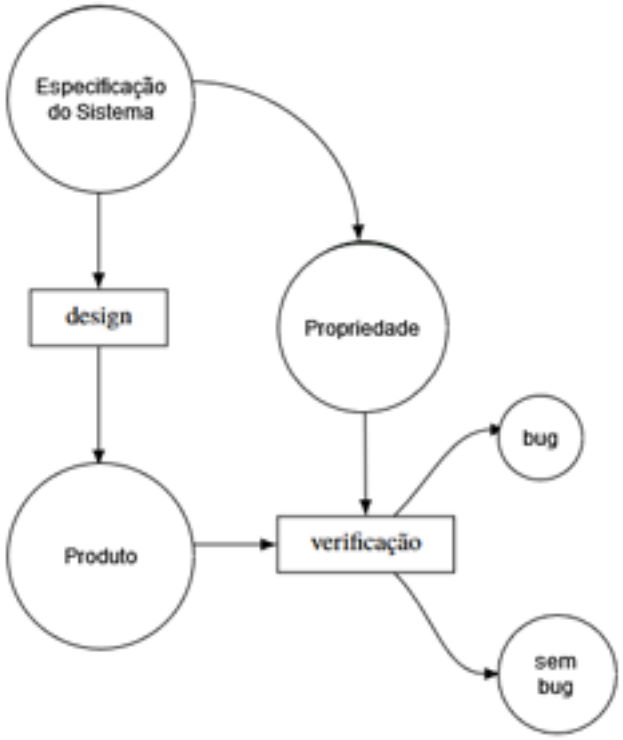

Fig. 1. Figura adaptada de Baier e Katoen(2008) [1]

3 PRÓS E CONTRAS

Dentro do que já consideramos da implementação, incluímos:

- `Prós`: Base sólida de matemática e lógica, aplicação acadêmica e industrial, geração de contra-exemplos, aplicável em diferentes contextos, verificação parcial.
- `Contras`: Caso as especificações não tenham sido definidas de maneira correta, os checadores de modelos podem apresentar erros. Caso a aplicação seja complexa demais, temos o problema da explosão dos estados.

3.1 Explosão de Estados

Mas afinal, qual a dimensão do problema da explosão de estados?

O gatilho desse problema reside na própria verificação do método, uma vez que para cada estado, todos os valores possíveis das variáveis existentes vão ser verificados, e isso ativamente demonstra, que esse crescimento vai ser exponencial, em Baier e Katoen(2008, p. 78) existe o exemplo do sistema de 10 estados, onde 8 variáveis conjuntas vão gerar um número possível de estados chegando a 8 milhões.

4 SISTEMAS DE TRANSIÇÃO

Um sistema de transição é definido como uma tupla (S, Act, →, I, AP, L) para a modelagem de sistemas computacionais, sendo a legenda:

- `S`: Conjunto de Estados.
- `Act`: Conjunto de ações.
- `I`: Conjunto de estados iniciais.
- `AP`: Proposições atômicas que contém info sobre diferentes estados do sistema(conclusão de uma tarefa).
- `L`: É a função que dita quais elementos de S atingem as proposições atômicas de AP.

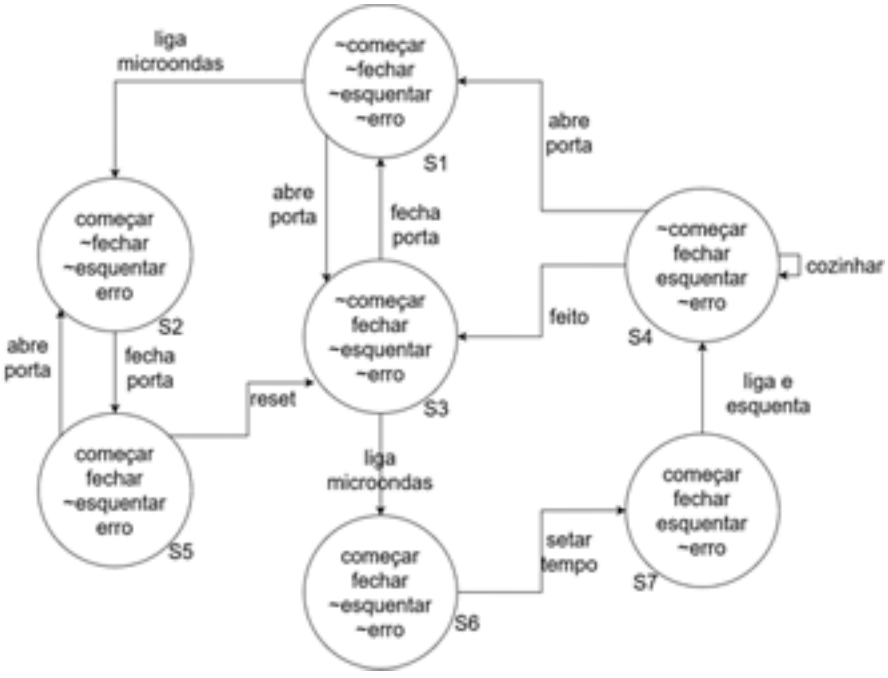

Fig. 2. Sistema de transição de um microondas

Um exemplo clássico de um sistema de transição, um sistema representando um microondas modelado pelo (S,Act,AP e I), onde: S é o conjunto de estados representados pelos nodos, *"Act"* é o conjunto de ações (começar,fechar,esquentar,erro), *"AP"* o conjunto de relações transtitivas e por fim I é q0.

5 PROPRIEDADES TEMPORAIS LINEARES

São as propriedades que descrevem os ponto desejados nas transições, linearmente. [3]

5.1 Segurança

- São propriedades utilizadas para garantir que nunca aconteça algum erro,operando corretamente
- Temos propriedades de segurança que tem de verificar em cada estado(invariantes) ou em variantes de caminhos finitos.
- Os melhores exemplos que temos disso são os deadlocks e casos de exclusão mútua.

5.2 Vivacidade(Starvation freedom)

- São propriedades definidas que demonstrem que o sistema execute uma função, produz algo.
- Dentro da propriedade de vivacidade, a garantia de que pelo menos 1 vez, o algoritmo tenha no seu conjunto "AP" de propriedades atômicas, pelo menos um estado de processo crítico.

5.3 Fairness

- São propriedades relacionadas ao comportamento como um todo do sistema.
- Dentro da propriedade de Fairness, temos que garantir de uma maneira justa, a forma como os processos participam no sistema, tentando evitar estados não-determinísticos em paralelo.
- A palavra-chave para essa propriedade é equidade e prioridade dentre processos.

6 LÓGICAS TEMPORAIS

Uma breve explicação das lógicas temporais empregadas na especificação de diferentes propriedades, estabelecendo a possibilidade de estabelecer sucessão temporal na ordenação dos processos. [2]

6.1 Lógica Temporal Linear (LTL)

- Assim como para a definição das propriedades lineares é necessário uma linguagem formal, uma das primeiras lógicas desenvolvidas para o mesmo fim é a lógica temporal linear.
- Utiliza-se os operadores clássicos da lógica proposicionale adiciona 2 operadores: o operador "próximo" (next,O) e "até" (until,U).
- Enquanto o operador next é unário, o operador until é binário.
- A limitação da LTL se encontra na necessidade de ter somente um sucessor temporal unitário.

6.2 Computation Tree Logic(CTL)

- A lógica da árvore herda os operadores da LTL , porém se difere na implementação pois assim como uma árvore, a lógica desta contém ramificações que permitem a utilização de propriedades especificadas para n caminhos possíveis.
- A especificação das propriedades de CTL é dividida em : estados e caminhos.

6.3 Temporal Computation Tree Logic(TCTL)

- A lógica de TCTL vem da necessidade da especificação de limites temporais em suas propriedades.
- Ela herda a mesma divisão de especificação de CTL: estados e caminhos.
- A lógica TCTL oferece a possibilidade da inclusão da possibilidade de N propriedade ser atingida em um intervalo I de tempo.

7 MODEL CHECKERS

Apresentado os conceitos, iremos apresentar 2 checadores de modelos frequentemente utilizados em aplicações.

7.1 SPIN

Trata-se de um Model Checker vastamente distribuído de maneira open source para verificação de sistemas, aparente em diferentes estudos, com exemplo:

- A verificação de um controlador de veículo espacial. [5]
- Sistema distribuído multi-threaded.

Sua modelagem deriva de uma linguagem textual (PROMELA - Process Meta Language), onde se modela o comportamento esperado do sistema e também disponibiliza uma análise sintática(Parser) onde erros são reportados. [7]

7.2 PRISM

Trata-se de um Model Checker probabilístico, multiplataforma utilizado em áreas como algorítimos quantícos, biologia e protocolos de comunicação, ele aceita como modelo cadeias de Markov, processos decisórios e autômatos temporais probabilísticos, também contém duas linguagens aceitas, uma para modelagem e a outra para especificação. [8]

- A linguagem textual PRISM com um conjunto de identificadores é utilizada para modelagem.
- A especificação é feita encima de uma linguagem específica PRISM para especificação de propriedades(capaz de lógicas LTL,CTL e PCTL).

8 CONCLUSÃO

A Ferramenta de checagem de modelos pode ser extremamente eficiente em oferecer uma validação matemática sólida de um sistema de modelos, condições a serem validadas e a vulnerabilidade na lógica do sistema/aplicação, além disso, também oferece diferentes opções de lógicas temporais, permitindo que as especificações do sistema também tenham uma orientação temporal utilizando operadores diferentes, assim como ramificações.

Porém isso não garante que sozinho, este método seja suficiente para todo tipo de aplicação. Como vimos na seção dos seus prós e contras, sistemas cada vez mais complexos vão deixando a ferramenta mais suscetível a ter dificuldade em ser ágil na sua verificação. O seu parser e sua lógica temporal oferecem ao software, um diagnóstico de problemas, em estado, execução, prioridade, ou mesmo na ambiguidade/ falta de precedência de diferentes especificações.

REFERENCES

[1] BAIER, C.; KATOEN, J.-P. Principles of model checking. Cambridge, Mass: The MIT Press, 2008. ISBN 978-0-262-02649-9 0-262-02649-X
[2] LAMPORT, L. What Good is Temporal Logic? Information Processing, v. 83, 1983.
[3] ALUR, R. Timed automata. In: Computer Aided Verification. Springer, 1999.

[4] KWIATKOWSKA, M.; NORMAN, G.; PARKER, D. PRISM 4.0: Verification of probabilistic real-time systems. In: Computer aided verification. Springer, 2011.
[5] HAVELUND, K.; LOWRY, M.; PENIX, J. Formal analysis of a space-craft controller using SPIN. IEEE Transactions on Software Engineering, v. 27, n. 8, 2001.
[6] STOLLER, S. D. Model-Checking Multi-threaded Distributed Java Programs. In: GOOS, G. et al. (Ed.). SPIN Model Checking and Software Verification. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.
[7] HOLZMANN, G. J. The model checker SPIN. IEEE Transactions on software engineering, n. 5, 1997.
[8] OXFORD, U. o. PRISM Manual | Main / Welcome. 2015. Disponível em: <http: //www.prismmodelchecker.org/manual/Main/Welcome>.